On Fairness in Simulatability-based Cryptographic Systems*

Michael Backes[1], Dennis Hofheinz[2], Jörn Müller-Quade[3], and Dominique Unruh[1]

[1] Saarland University, Germany, {backes,unruh}@cs.uni-sb.de
[2] CWI, Amsterdam, Dennis.Hofheinz@cwi.nl
[3] IAKS, Universität Karlsruhe, muellerq@ira.uka.de

Abstract. Simulatability constitutes the cryptographic notion of a secure refinement and has asserted its position as one of the fundamental concepts of modern cryptography. Although simulatability carefully captures that a distributed protocol does not behave any worse than an ideal specification, it however does not capture any form of liveness guarantees, i.e., that something good eventually happens in the protocol.

We show how one can extend the notion of simulatability to comprise liveness guarantees by imposing specific fairness constraints on the adversary. As the common notion of fairness based on infinite runs and eventual message delivery is not suited for reasoning about polynomial-time, cryptographic systems, we propose a new definition of fairness that enforces the delivery of messages after a polynomial number of steps. We provide strengthened variants of this definition by granting the protocol parties explicit guarantees on the maximum delay of messages. The variants thus capture fairness with explicit timeout signals, and we further distinguish between fairness with local timeouts and fairness with global timeouts.

We compare the resulting notions of fair simulatability, and provide separating examples that help to classify the strengths of the definitions and that show that the different definitions of fairness imply different variants of simulatability.

Keywords: fairness, simulatability, cryptographic protocols, scheduling.

1 Introduction

Simulatability constitutes the cryptographic notion of a secure refinement and has asserted its position as one of the fundamental concepts of modern cryptography. Although simulatability carefully captures that a distributed protocol does not behave any worse than an ideal specification, it however does not capture any form of liveness guarantees, i.e., that the protocol ensures that something good eventually happens. As a consequence, protocols are considered to be secure even if a single corrupted player can prevent the protocol from terminating. Clearly, this can lead to unsatisfactory situations, especially in protocols in which liveness aspects are considered crucial, e.g., in an electronic voting scheme.

One solution to this is to explicitly check protocols for liveness properties. This approach has the drawback that such properties have to be formulated individually for each and every protocol task. Furthermore, it is unclear how such explicitly formulated properties behave under protocol composition. In this paper, we investigate how one can extend the notion of simulatability itself so that it comprises liveness guarantees. The natural solutions as well as the one we choose in this paper is to restrict the master scheduler—the adversary in our case—to fair scheduling.

* This is the full version of [BHMQU05] which appeared at the FMSE'05, Alexandria. Research was done while the first author was at IBM Research, Zurich and the second and fourth author were at the IAKS, Universität Karlsruhe.

However, the common definition of fairness based on infinite runs and eventual message delivery is not suited for reasoning about cryptographic systems whose parties are required to run in polynomial-time. Hence we first define a new notion of fairness corresponding to a polynomial-time variant of the usual fairness definition, i.e., we require that every message be scheduled within a specific, polynomially bounded number of steps of the adversary instead of requiring eventual delivery of every message.

The new notion of fairness guarantees protocol participants that their messages are delivered, but as the specific polynomial need not be known to the participants, they cannot decide whether a message has been sent at a particular time or within a particular time interval. This is in contrast to practical scenarios where timeouts are usually explicitly used to avoid (or attenuate) situations where a corrupted protocol participant can prevent a protocol from terminating, thereby granting the participants additional capabilities of continuing with a protocol. It hence seems promising to extend the new definition of fairness with explicit timeouts and to compare the strength of the resulting notions in simulatability proofs. We thus provide strengthened variants of polynomial-time fairness that we call fairness with timeouts, which make the guaranteed maximal delay times of the messages known to the participants. Knowing the delay times will allow the protocol participants to distinguish between a message that is delayed by the network and a message that was not sent at all. We will distinguish between two variants of fairness with timeouts: First, there is fairness with local timeouts, which provides different delay times for different connections, and each participant learns only the guarantees of its own connections. Second, there is fairness with global timeouts, which provides a globally unique delay time for each connection, and each participant learns this time.

We compare the definitions of fair simulatability resulting from the different notions of fairness, and we provide separating examples that help classify the strengths of the simulatability definitions. One would be tempted to think that a protocol that is secure with respect to one definition will be secure with respect to a definition that provides more comprehensive fairness guarantees. However, and somewhat counterintuitively, the examples have shown that this is not always the case. This stems from the fact that simulatability is defined by comparing a real protocol with an ideal specification. Hence the more guarantees are given in the ideal model, the more requirements have to be fulfilled by the real protocol. In a nutshell, we show that our different definitions of fairness imply different definitions of fair simulatability. The separations shown in this work are not given by protocols that are secure in one network model and become insecure in another network model, but by protocol tasks that can be realized with respect to one scheduling and cannot in principle be realized with respect to another. More specifically, we show that a specification of broadcast protocols can be securely realized in a nontrivial manner with respect to usual (nonfair) simulatability, but that broadcast cannot be securely realized with respect to fair simulatability. Moreover, we prove that there is a simple and intuitive protocol task that separates fair simulatability and fair simulatability with timeouts. Finally, we show that there is a simple and intuitive protocol task that separates fair simulatability with global timeouts and fair simulatability with local timeouts.

1.1 Related Work

Simulation-based definitions of security were given for the synchronous model [PSW00, Can00] and the asynchronous model [PW01, Can01]. In [BPSW02] scheduling with fairness properties is introduced to prove liveness properties. This scheduling of [BPSW02] differs from the fair scheduling presented here in that guarantees are given only for service ports, and the adversary can be stopped by the user to let all waiting messages be delivered, thus ensuring liveness.

In [Bac03] a construction was introduced that allows synchronous protocols to be represented in an asynchronous network such that asynchronous security with respect to the new representation implies security in the synchronous model. This result holds with respect to the specific representation used and does not imply a relation between asynchronous and synchronous security. The work [HMQ04] introduces timeout-fair scheduling (which is called "reliable scheduling" there) in a model of security specifically designed for this purpose. For the timeout-fair scheduling it is proved that oblivious transfer (together with broadcast) is not complete.

Timing issues for non-simulation-based definitions of security have long been studied in cryptography. Fault tolerance has been studied primarily in connection with agreement and consensus problems. A task that is impossible in a completely asynchronous network, but possible with synchronous communication, was given in [FLP83]. In [DLS88] this impossibility result was studied in a network where delivery is guaranteed, but where the bounds for possible delays are not known to the protocol participants. This network model is adapted to the security model of [PW01] in this work by the use of fair schedulers. In [ADG84] an asynchronous model is used that gives delivery guarantees for messages sent by non-faulty processors. These guaranteed maximal delay times are known to all participants, and we adapted this type of scheduling to the security model of [PW01] under the name of globally reliable scheduling. A fair scheduling where the adversary can suppress messages only with a certain probability is defined in [BT85], but if one does not consider efficiency this network model can be made reliable by sending messages multiple times. In more recent work [CKPS01] a machine sends messages to itself to measure time. A similar approach is used here to implement timers by self-loops and delivery guarantees.

The study of more general cryptographic protocols in an asynchronous setting was initiated by [BOCG93]. Differences between asynchronous and synchronous scheduling were shown in a simulation-based security model. Another work relating to fairness in the context of simulation-based security is [GMY04]. However, fairness is denoted there as the property where no party has an advantage at the end of the computation. The underlying network model is synchronous. In the context of proactive security, asynchronous networks are investigated in [CKLS02]. In a completely asynchronous model, proactive security is shown to be impossible, but with some synchronization it becomes possible, thereby showing an influence of scheduling on security.

1.2 Overview

In Section 2, we briefly summarize the model of security used here. Section 3 first motivates and then rigorously defines the notion of a fair scheduler. Section 5 introduces two variants of fair schedulers. In Section 6, we investigate the relationships among our new security notions and existing ones. The paper concludes with Section 7.

2 Reactive Simulatability

Our work is based on the model of reactive simulatability [PW01, BPW07], which is an asynchronous probabilistic execution model with distributed scheduling that provides universal composability properties while including computational aspects as needed for cryptography. The model is automata based, i.e., protocols are executed by interacting machines, and event-based, i.e., machines react on certain inputs. All details of the model that are not necessary for understanding are omitted; they can be found in the original papers.

In particular, we repeat the scheduling model in detail because it is important for the definitions of fairness. The specific scheduling aspects needed for cryptographic asynchronous systems are that schedulers are "normal" system machines so that they schedule with realistic knowledge, and that different channels may be scheduled by different machines, e.g., so that local submachines can be represented.

2.1 General System Model

A *machine* is a probabilistic IO automaton (extended finite-state machine) in a slightly refined model to allow complexity considerations. For these automata, Turing-machine realizations are defined, and the complexity thereof is measured in terms of a common security parameter k, given as the initial work-tape content of every machine. A *structure* consists of a set $\hat{M}$ of connected machines (also called a *collection* henceforth) and a subset S of free *ports*, called *service ports*. Each structure is complemented to a *configuration* by a *user* machine H, modeling the entirety of the honest users, and an *adversary* machine A. The machine H connects only to ports in S, whereas A connects to the remaining free ports of the structure and may interact with the users. We denote the set of configurations of a structure $(\hat{M}, S)$ by $\mathsf{Conf}(\hat{M}, S)$ and the subset of polynomial-time configurations by $\mathsf{Conf}_{\mathsf{poly}}(\hat{M}, S)$. A *protocol* (called a *system* in [PW01, BPW07]) is then modelled as a family of structures $(\mathsf{struct}_A)_A$ where A ranges over all possible sets of uncorrupted parties. (I.e., in a protocol where at least t out of n parties are guaranteed to be honest, A would range over all sets of parties with $|A| \geq t$.) Each structure struct_A then models the original protocol with all machines corresponding to parties not in A removed (such that the adversary may take over all connections that where owned by these machines). Since a system will be defined to be secure if each of its structures is secure, in the following we will for simplicity only consider the security of structures.[4] For details we refer the reader to [BPW07].

The general scheduling model in [PW01, BPW07] gives each connection q (from an *out-port* $\mathsf{q}!$ to an *in-port* $\mathsf{q}?$) a *buffer* $\widetilde{\mathsf{q}}$, and the machine with the corresponding *clock out-port* $\mathsf{q}^{\triangleleft}!$ can schedule a message there when it makes a transition, cf. Figure 1 (note that some or all of these ports may belong to the same machine). Scheduling of machines is done sequentially, so there is exactly one active machine M at any time. The machine receives messages at its in-ports (representing incoming network connections) and may output messages at its out-ports. An output message is appended to a queue of messages maintained by the buffer associated with the respective out-port. If the active machine has clock out-ports, it can select the next message to be scheduled by outputting a number $n \geq 1$ to one clock out-port $\mathsf{q}^{\triangleleft}!$. If the buffer $\widetilde{\mathsf{q}}$ contains at least n elements, the n-th message of buffer $\widetilde{\mathsf{q}}$ is delivered to the unique receiving machines that has the port $\mathsf{q}?$, and the message is removed from the buffer. The unique receiving machines becomes the next active machine. If M tries to schedule multiple messages, only one is taken, and if it schedules none, if the message does not exist, and at the start of the run, the special *master scheduler* is scheduled. In our setting, we assume the adversary to be the master scheduler. Usually, a connection is clocked by (i.e., the corresponding clock out-port is part of) the sender (a delay-less connection), or by the adversary (an asynchronous connection). For simplicity, we disallow a machine to clock a connection between two other machines in a structure (which does not have a natural counterpart in the real world). The most important use of a clock out-port is to schedule the oldest (and typically only) message in a buffer, i.e., to output 1 at the respective clock out-port. We then say that the machine *schedules* the buffer or the connection.

This means that a closed collection, i.e., a collection whose ports are fully connected, has a well-defined notion of *runs*, also called *traces* or *executions*. Formally a run is essentially a sequence of *steps*, and each step is a tuple of the name of the active machine in this step and

[4] With the exception of Section 4 where a special treatment is necessary for the case with no corrupted party is necessary.

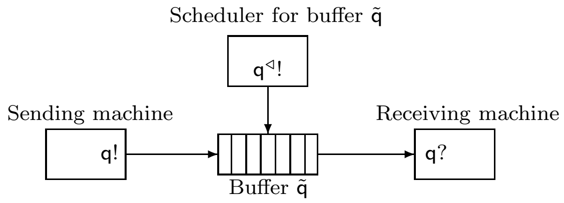


Fig. 1. Ports and Buffers

its input, output, and old and new local state. As the underlying state-transition functions of the individual machines are probabilistic, we also get a probability space on the possible runs. We call it $run_{\hat{C},k}$ for a collection $\hat{C}$ and the security parameter k. One can restrict a run r to a machine M or a set of machines $\hat{M}$ by retaining only the steps of these machines; this is called the *view* of these machines, and the corresponding random variables are denoted by $view_{\hat{C},k}(\mathsf{M})$ and $view_{\hat{C},k}(\hat{M})$, respectively. For a configuration $conf = (\hat{M}, S, \mathsf{H}, \mathsf{A})$ we simply write $run_{conf,k}$ instead of $run_{\hat{M}\cup\{\mathsf{H},\mathsf{A}\},k}$, and similar for views.

2.2 Reactive Simulatability

Simulatability constitutes the cryptographic notion of secure implementation and has asserted its position as a fundamental concept of modern cryptography. For reactive systems, it means that whatever might happen to an honest user in a (typically real) structure $(\hat{M}_1, S)$ can also happen in a (typically more ideal) structure $(\hat{M}_2, S)$ given as a specification: For every user H and every real adversary A_1 of the real structure, there exists an ideal adversary A_2 (also called simulator) such that the views of H are indistinguishable if H is either run with the real structure and the real adversary, or with the ideal structure and the simulator. This is illustrated in Figure 2. The most important notion of indistinguishability is called computational indistinguishability, which is a well-known cryptographic notion from [Yao82] that captures that two (families of) random variables cannot be distinguished in probabilistic polynomial time. Other common notions of indistinguishability are perfect indistinguishability ("$\approx_{\mathsf{perf}}$"), which requires the families to be identical, and statistical indistinguishability ("$\approx_{SMALL}$"), which requires the statistical distance of the families to be a function of a class $SMALL$.

Definition 1 (Reactive Simulatability). *For two structures $(\hat{M}_1, S)$ and $(\hat{M}_2, S)$ with identical sets of service ports, and $x \in \{\mathsf{perf}, SMALL, \mathsf{poly}\}$, one says $(\hat{M}_1, S) \geq^x_{\mathsf{sec}} (\hat{M}_2, S)$ (*at least as secure as*) iff for every configuration $conf_1 = (\hat{M}_1, S, \mathsf{H}, \mathsf{A}_1) \in \mathsf{Conf}(\hat{M}_1, S)$, there exists a configuration $conf_2 = (\hat{M}_2, S, \mathsf{H}, \mathsf{A}_2) \in \mathsf{Conf}(\hat{M}_2, S_2)$ (with the same H) such that $view_{conf_1}(\mathsf{H}) \approx_x view_{conf_2}(\mathsf{H})$. In the case $x = \mathsf{poly}$, H, A_1, and A_2 have to be polynomial-time.*

For $x = \mathsf{poly}$ we speak of computational, *for $x = SMALL$ of* statistical, *and for $x = \mathsf{perf}$ of* perfect reactive simulatability. *We write $\geq_{\mathsf{sec}}$ if x is clear from the context and speak of* reactive simulatability. Universal *simulatability, written $\geq^{\mathsf{univ}}_{\mathsf{sec}}$, means that A_2 does not depend on H (only on $\hat{M}_1$, S, and A_1).*

In the following, we will sometimes call this notion *asynchronous* reactive simulatability to distinguish it from the other notions introduced below.

An essential feature of this definition of simulatability is a composition theorem [PW01, BPW07], that roughly says the following: Given a structure A (usually a protocol) that is at

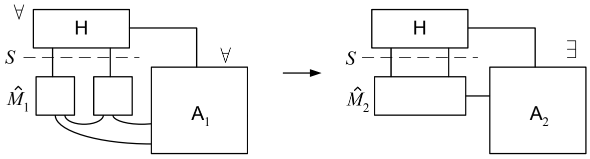


Fig. 2. Simulatability example: The two views of H must be indistinguishable

least as secure as a structure B (usually some primitive), and given a protocol X^B (having B as a sub-protocol, i.e., using the primitive), the protocol X^A obtained by replacing B by A in X^B is at least as secure as X^B. This allows to modularly design protocols, i.e., one first designs the protocol X^B and then proceeds by deriving an implementation for B that is secure in the sense of reactive simulatability.

3 Fairness in Simulatability-based Cryptographic Systems

Albeit being a powerful notion for establishing the security of cryptographic tasks, the notion of reactive simulatability does not provide any assurance that messages are in fact delivered (except if immediate delivery of message is desired and explicitly modeled, which would constitute too strong an assumption in most cases). More precisely, a protocol that does not produce any outputs is at least as secure as *any* other protocol. In other words, reactive simulatability does not enforce liveness.

The common solution to achieve liveness is by relying on a fair scheduler. In our scenarios, this corresponds to requiring the adversary as the master scheduler to eventually deliver all messages. However, as already stated in the introduction, a definition of fairness that is suitable for reasoning about cryptographic systems has to take computational restrictions into account; in particular, this stands in contrast to the traditional notion of eventual delivery of messages which is based on runs of infinite length. Once a suitable notion of fairness for adversaries is in place, we can then restrict the simulatability definition to the class of fair adversaries.

3.1 Fair Schedulers

In contrast to the traditional notion of fairness—any message sent over the network will eventually arrive—a concise treatment of fairness in the presence of cryptography imposes several additional difficulties.

First, delivery should happen after a polynomial number of steps. Otherwise, a scheduler may suppress message delivery until all protocol machines have halted. We will therefore require the existence of a polynomial F that bounds the number of activations of the scheduler between two clockings of any connection in the security parameter.

Secondly, we explicitly have to exclude schedulers that stop working, e.g., because they reach a final state. This would relieve them of their duty to schedule the network connections fairly. In particular, this excludes machines that are polynomially bounded in the traditional sense, i.e., those that halt after a polynomial number of overall steps.

Since this excludes the use of the usual definition of computational security in our setting, we need a different notion of computational security where the adversaries are not required to eventually terminate. To allow for a sensible notion of fairness, we therefore consider a refined notion of polynomial-time users and adversaries here, following ideas initiated in [HMQU05]. For describing this refinement, let us first review the intuitive idea underlying computational security.

Computational security states that a security property is maintained unless the adversary has super-polynomial power. To capture this idea, it is sufficient to assume that participants in a communication do not have immense computational power *per time unit*. We do not care whether they may break any hard problem when computing an exponential amount of time, since we only consider events (like breaking the protocol) which happen in a conceivable future, e.g., not after 10^{10} years.

In other words, we drop the hard polynomial bound on the overall number of steps a machine may perform, but instead we bound the machines only in such a way that in polynomial prefixes of the users' view, the overall number of steps of all machines is polynomial. Consequently, we only consider polynomial prefixes of the users' view for security comparisons, hence it suffices to restrict H to be polynomial-time *in each activation*. Moreover, the adversary A must be kept polynomial in the size of H's view. Thus, we demand that A is polynomial in the overall size of all inputs from H and outputs A gave to H. However, it should be stressed that neither H nor A actually halts. In particular, A cannot delay message delivery up to a point in time where H would not see the consequences of this delivery.

Such users and adversaries are called *continuously polynomial*. The corresponding security notion, i.e., the restriction of reactive simulatability to continuously polynomial users and adversaries, behaves well under composition and is stricter than the original notion of reactive simulatability. For detailed proofs of these claims and rigorous definitions, we refer to [HMQU05]. Here, it is important that we can use this notion to sensibly catch what it means for an adversary to be a fair scheduler.

Definition 2 (Fair Schedulers). *Let* M *be a machine,* $\mathsf{p}^{\triangleleft}!$ *a clock out-port of* M*, and* $F\colon \mathbb{N}_0 \to \mathbb{N}_{>0}$ *a function. We say that* M F-schedules $\mathsf{p}^{\triangleleft}!$ *if in every closed collection* $\hat{C}$ *that contains* M*, and for every sufficiently large security parameter* $k \in \mathbb{N}$*,* M *schedules the port* $\mathsf{p}^{\triangleleft}!$ *at least every* $F(k)$*-th activation.*

The machine M *is* F-fair *if it is a master scheduler that never halts and* F*-schedules every of its clock out-ports. If* F *is a polynomial,* M *is called* polynomially fair*, or simply* fair*. Continuously polynomial users / adversaries which are fair are called* computationally fair users / adversaries*.*

3.2 Fair Reactive Simulatability

The restriction of reactive simulatability to (computationally) fair users and adversaries now yields the notion of *fair reactive simulatability*.

Definition 3 (Fair Reactive Simulatability). *Let* $(\hat{M}_1, S)$ *and* $(\hat{M}_2, S)$ *be structures, and let further* $x \in \{\mathsf{perf}, \mathit{SMALL}, \mathsf{poly}\}$*. We call* $(\hat{M}_1, S)$ at least as secure as $(\hat{M}_2, S)$ with respect to fair adversaries *(written* $\geq_{\mathsf{sec}}^{x,\mathsf{fair}}$*) iff for every configuration* $\mathit{conf}_1 = (\hat{M}_1, S, \mathsf{H}, \mathsf{A}_1)$ *with fair adversary* A_1*, there exists a configuration* $\mathit{conf}_2 = (\hat{M}_1, S, \mathsf{H}, \mathsf{A}_2)$ *with fair adversary* A_2 *such that* $\mathit{view}_{\mathit{conf}_1}(\mathsf{H}) \approx_x \mathit{view}_{\mathit{conf}_2}(\mathsf{H})$*. In the case* $x = \mathsf{poly}$*,* A_1 *and* A_2 *have to be computationally fair, and* H *has to be continuously polynomial. Universal fair simulatability is defined in an analogous manner.*

We only briefly note that Definition 3 behaves well under composition. In addition to the composability proof of the original notion of reactive simulatability, we have to investigate aspects of fairness and of continuous polynomial-time. It has been shown in [HMQU05] that polynomially bounded protocols can be composed without losing continuously polynomial security. The proof was conducted by reducing an attack on the composed protocol to an attack against a subprotocol. Since the subprotocols are secure by assumption, there is a corresponding ideal adversary

A_2 which is then shown to be a good simulator for the attack on the composed protocol. For the definition of fair reactive simulatability, the same proof applies if one additionally shows that the constructed simulator A_2 is fair. This can easily be established since the notion of fairness of an adversary does not depend on the protocol it is run with.

4 Non-Trivial Protocols

The notion of fair reactive simulatability as introduced in the preceding section allows us to capture the idea of protocols that eventually terminate. This gives rise to the question whether the requirement of eventual termination will reduce the number of realisable cryptographic tasks. More concretely, is there a cryptographic task (i.e., a trusted host, a primitive, or a functionality) that can be realised asynchronously but not with respect to fair adversaries. It turns out that this is the case due to a mere technicality: Any cryptographic task[5] is realised by the trivial protocol that never gives any output and never sends any messages. Obviously, this is not the case for fair reactive simulatability. So why does the trivial protocol asynchronously realise any functionality? Assume any adversary. Whatever this adversary does, the trivial protocol will never send any message to the environment. The simulator can easily reproduce this behaviour by never scheduling any of the outputs of the ideal functionality. Thus we have security in the sense of asynchronous reactive simulatability.

This problem was first addressed in [Can01] by requiring a protocol to be non-trivial. A natural way to define non-triviality would be to require the protocol to eventually generate output as long as no party is corrupted (as was done in [Can01]). However, this approach leads to the following problems:

- Consider a trusted host that may—even when used in a honest way—generate no output. For example, a functionality for password authenticated key exchange may be formalised to output a key if the passwords match and to be silent otherwise. A protocol implementing that functionality will necessarily violate the requirement of eventual generation of output (at least for some inputs).
- To avoid this problem, one might choose to define a protocol as non-trivial if there exists some honest user such that for all adversaries the protocol will eventually give output. However, consider the functionality for authenticated message transmission. This functionality is implemented by a protocol that silently ignores all messages except for strings consisting only of 0's. This protocol may then asynchronously realise the functionality. Since there exists an honest user that sends only such strings, we would further call such a protocol non-trivial. However, such a protocol is clearly not a satisfying implementation of an authenticated channel.
- Another question is whether we require that *all* protocol parties give output or that *at least one* party gives output. In the first case, a protocol for secure message transmission in which only the recipient has output would not be considered non-trivial. If we choose the second variant, we can non-trivially implement the coin-toss functionality using the following simple protocol: Upon the first activation, Alice outputs a random bit. Bob never gives output. This protocol is an asynchronously secure implementation of coin-toss since the simulator may choose never to deliver the answer on Bob's side of the functionality. However, this protocol is non-trivial if we only require *one* party to give output. Of course, such a one-sided non-trivial protocol does not fulfil the intuitive requirement of non-triviality, in particular in view of the fact that a non-trivial implementation of coin-toss (in an intuitive sense) has been shown impossible by [CKL03].

[5] At least if it is not given as a *localised* functionality.

– Finally, even if we capture the intuitive meaning of non-triviality, it is not guaranteed that this notion composes securely. That is, we might have a non-trivial secure implementation π of a given task, and we replace some primitive used by π by some non-trivial subprotocol, we may ask whether the resulting protocol is still non-trivial. If possible, a definition of non-triviality should fulfil this condition.

These arguments show that the actual meaning of non-triviality seems to depend on the actual functionality that is being implemented. Thus one would have to give a separate definition of non-triviality for each and every cryptographic primitive. However, this contradicts the idea of simulatability where the specification of a protocol task is given by the functionality alone. Instead, a functionality would have to come paired with a non-triviality specification. Fortunately, there is a better solution to this problem. We call a protocol $\hat{M}_1$ non-trivial with respect to some ideal protocol $\hat{M}_2$ if, given that no party is corrupted and when considering only adversaries that deliver all messages, $\hat{M}_1$ securely realises $\hat{M}_2$.[6] Somewhat more formally, $\hat{M}_1$ is non-trivally asynchronously as secure as $\hat{M}_2$ if $\hat{M}_1$ is asynchronously as secure as $\hat{M}_2$ and additionally $\hat{M}_1$ is as secure as $\hat{M}_2$ *with respect to fair adversaries* when we do not allow corruptions. Formally, this is defined as follows:

Definition 4 (Non-trivial asynchronous reactive simulatability). *Let π and ρ be two protocols (systems in the nomenclature of [PW01, BPW07]). Let A_0 be the set of all protocol parties. Assume that π and ρ consist of structures $(\mathsf{struct}^{\pi}_{A})_{A\in\mathfrak{A}}$ and $(\mathsf{struct}^{\rho}_{A})_{A\in\mathfrak{A}}$, respectively, where A ranges over sets $A \in \mathfrak{A}$ of uncorrupted parties. Assume that $A_0 \in \mathfrak{A}$.*

We then call π non-trivially asynchronously as secure as *ρ (written $\geq^{x,\mathsf{nt}}_{\mathsf{sec}}$) iff the following two conditions are fulfilled:*

– *The structure $\mathsf{struct}^{\pi}_{A_0}$ is as secure as $\mathsf{struct}^{\rho}_{A_0}$ with respect to fair adversaries (i.e., $\mathsf{struct}^{\pi}_{A_0} \geq^{x,\mathsf{fair}}_{\mathsf{sec}} \mathsf{struct}^{\rho}_{A_0}$).*
– *For every $A \in \mathfrak{A}$, we have that $\mathsf{struct}^{\pi}_{A}$ is asynchronously as secure as $\mathsf{struct}^{\rho}_{A}$ (i.e., $\mathsf{struct}^{\pi}_{A} \geq^{x}_{\mathsf{sec}} \mathsf{struct}^{\rho}_{A}$).*

For this notion, composability now directly follows from the fact that both asynchronous reactive simulatability and fair reactive simulatability compose securely.

Obviously, non-trivial asynchronous reactive simulatability is at least as strict as asynchronous reactive simulatability. In [CKL03] is has been shown that the coin-toss functionality can be asynchronously realised by the trivial protocol but cannot be realised non-trivially, not even in the case of polynomial reactive simulatability. (Their proof used a notion of non-triviality that was tailored for the case of coin-toss. However, their proof easily adapts to our definition.)

We can now restate the question from the beginning of this section: Is there a cryptographic task (i.e., a trusted host, a functionality) that can be realised statistically non-trivially but that cannot be realised with respect to fair adversaries? This question is answered positively in Section 6.1.

5 Variants of Fairness with Timeouts

In Section 3 we have elaborated on the benefits of protocols that eventually terminate. Many practical protocols will however only provide a guarantee of eventual termination if timeout signals are used appropriately. A mail server trying to deliver mail will not forever try to talk to

[6] This approach was already pursued in [CLOS02]. However, they gave no definition of adversaries that deliver all message (non-blocking in their nomenclature). As the present paper shows, such a definition is far from straightforward.

another server but will after some time either try another server or abort with an error message; a computer that auto-detects printers in a network will, after waiting a given amount of time, stop and consider the list as complete; an election protocol may exclude voters that do not vote within a specified time frame as is common in the conventional election method using the non-electronic ballot-and-urn method. This exemplifies the need of suitably capturing timeouts as well in simulatability-based cryptographic systems.

We first discuss what a protocol must have at its disposal to implement timeouts. First and foremost, there should be a means of measuring time. Additionally, there should be some guarantees concerning the time needed for a message to be delivered. If no such guarantees exist, implementing a timeout would risk ignoring messages from uncorrupted parties since their connections might delay messages beyond the chosen timeout.

When expressing these two concepts, we tried to use as few additional assumptions as possible; in particular, we did not want to imply that different machines had synchronous clocks or even only clocks running at the same speed. We tried to meet this condition by capturing the possibility of measuring time by introducing so-called *time lines*. These are special designated connections, usually self-loops, that guarantee that messages on these connections are never delivered too fast. Time lines can be used to measure time since after n clockings of a given time line, at least n times a given amount of time (say n "seconds") passed. However, clocking of time lines can take place much less frequently, hence only very weak synchronization among different parties can be realized using time lines. In real-world implementations, time lines are naturally realizable by normal clocks, and one would simply assume that time lines deliver, e.g., one message per second. To capture the notion of time lines formally in the model, we assign a specific prefix $\mathsf{time\_}$ to the names of the respective ports, i.e., connections are considered as time lines if the names of the respective ports start with $\mathsf{time\_}$. We call such ports *time ports*.

We furthermore have to implement guarantees on the maximum delay of a given connection. We achieve this by forcing the adversary to send a number $J(k)$ to some or all parties (before any other machine is activated). The adversary is then obliged to clock any time line at most $J(k)$ times between two clockings of any connection. This allows any party to realize a timeout for any given connection by waiting for $J(k)$ clockings of its time line (i.e., $J(k)$ "seconds") before assuming the message to be delivered. To prevent the adversary from choosing arbitrary large values $J(k)$, we require J to be a fixed function that is polynomially bounded in the security parameter. $J(k)$ hence serves as an a priori and generally known upper bound on the delay of a connection. Such upper bounds are usually known in practice, at least if the hardware used in the protocol is known (we may have to use quite generous bounds to be sure). Similar to time lines, we assign the names of ports on which $J(k)$ is to be sent by the adversary a prefix $\mathsf{fair\_}$. We call such ports *guarantee ports*.

5.1 Fairness with Global Timeouts

Combining the notions of time lines and of guaranteed maximum delay of a connection for all users with the notion of fairness in the sense of Definition 2 yields the following variant of fairness, which we call *fairness with global timeouts*. We speak of global timeout to distinguish them from so-called local timeout that provide a guaranteed maximum delay only for some distinguished connections and as such constitutes only a local guarantee. We will address local timeouts in Section 5.2.

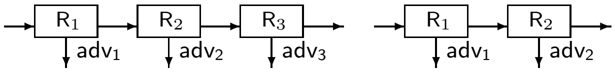
Definition 5 (Fair Schedulers with Global Timeouts). *Let* M *be a machine and* $J : \mathbb{N}_0 \to \mathbb{N}_{>0}$ *a function. We say that* M *is* fair with global timeouts of delay J, *if*

– *The machine* M *is fair.*

– *For any clock out-port* $\mathsf{p}^{\triangleleft}!$ *and any time port* $\mathsf{t}^{\triangleleft}!$ *of* M*, any closed collection* $\hat{C}$ *containing* M*, the following holds (with probability one over the runs of* $\hat{C}$*): The machine* M *does not schedule the port* $\mathsf{t}^{\triangleleft}!$ *more than* $J(k)$ *times without scheduling* $\mathsf{p}^{\triangleleft}!$ *at least once.*
– *In its first activation,* M *writes* $J(k)$ *(in unary representation) to all guarantee out-ports. Furthermore,* M *never writes anything else to the guarantee out-ports.*

A machine M *is called* fair with global timeouts *if it is fair with global timeouts of delay* J *for some polynomially bounded* J*.*

Extending the definition of fair reactive simulatability to comprise global timeouts can be derived as usual by considering fair adversaries with global timeouts instead of only fair adversaries. We refer to this notion by $\geq_{\mathsf{sec}}^{\mathsf{gtfair}}$ in the following. It can easily be shown that $\geq_{\mathsf{sec}}^{\mathsf{gtfair}}$ retains compositionality; the proof can be conducted along the lines of the original compositionality proof for reactive simulatability and its extension to fair adversaries.



(a) (b)

Fig. 3. Chains of repeaters

The notion of fair reactive simulatability with global timeouts allows us to specify and examine protocols using timeouts. However, it turns out that fair reactive simulatability with global timeouts constitutes a rather strict notion. Consider the following construct: Let R_i be machines that take an input of length k on $\mathsf{in}?$, forward it to $\mathsf{out}!$, and copy it to the adversary via $\mathsf{adv}_i!$. Assume three such machines to be connected to form a protocol $\hat{M}_1$ as in Figure 3 (a) (with ports renamed accordingly). Compare this protocol with $\hat{M}_2$ which consists of only two such repeaters as shown in Figure 3 (b). Intuitively, we would assume three repeaters to implement two repeaters, at least in a world where exact time measurements are not possible. However, this is not the case: Assume that a real adversary schedules the connections between the repeaters as seldom as possible (i.e., every $J(k)$-th time line clocking). Thus an honest user H with a time line will be able to deduce that a message sent through the three repeaters has a round trip time of $4J(k)$ time line clockings. An ideal adversary A_2 now has to guarantee the same $J(k)$ to avoid being distinguishable by the announced $J(k)$ in a trivial manner. So A_2 can deliver a message through the two repeaters no slower than within $3J(k)$ time-line clockings, which gives distinguishability. So $\hat{M}_1$ is not as secure as $\hat{M}_2$.

What is the impact of this observation? Any natural specification of a trusted host will have response times which are fixed and small multiples of $J(k)$ (e.g., delay of the in-port, delay of the out-port, and delay of some self loop giving the adversary time to modify the result, giving $3J(k)$). Complex protocols however take a large number of communication steps, thus having a much larger response time. Then using similar arguments as with the repeaters, one can see that such a complex protocol can never be as secure as the simple trusted host. However, designing the trusted host to have so much delay that the protocol can still be implemented would seem unnatural, since an ideal trusted host should abstract from protocol implementation details.

Several solutions come to mind. We could forbid the honest user to have time or guarantee ports and thus prevent it from noticing the complexity difference between the protocols. Then however the composition theorem would not hold any more since its proof makes use of the fact that protocol parties are combined into the honest user without changing the behavior of the overall network (this would now be impossible for parties having time lines and guarantee ports). In fact, the proof does not only become invalid but counterexamples can easily be constructed.

Alternatively, we could free the simulator from the obligation of fulfilling the guarantees he gave. This seems reasonable at first glance since the added guarantees only allow the use of timeouts in the real-life protocol. However, when imposing less restrictions on the simulator than on the adversary, the notion of fair reactive simulatability with global timeouts would not be transitive any more, and hence the composition theorem would not be very useful.[7]

One way to circumvent this is to have a canonical way to "slow down" ideal connections by inserting special buffers that need to be clocked a certain, fixed number of times to deliver. This still allows for reliability in the sense that ideal messages are delivered after a polynomial number of steps. However, the concrete reliability is in general not known to the ideal host, since the delaying buffers (or, *delay boxes*) are inserted *after* specification of the ideal host. This method is explored in detail in Appendix A. Another way of catching the notion of reliable communication lines is presented in the next section.

5.2 Fairness with Local Timeouts

The notion of fairness with global timeouts requires the adversary to give global delivery guarantees, i.e., guarantees that are valid for all of the adversary's clock out-ports. We have seen that simulatability problems arise out of the fact that these guarantees have to be identical in the real and ideal settings, and we showed that these problems can be suitably tackled by delaying ideal structures. However, these problems do not even arise when considering only local timeouts corresponding to local delivery guarantees. Local delivery guarantees are scheduling guarantees which relate only to a specific connection, i.e., a guarantee is only given to a machine that is sender or receiver of the considered connection.

Local timeouts can be motivated and justified by the situation of a very large protocol where it seems plausible to assume that each protocol participant knows delivery guarantees for its local connections. For example, the hardware used for direct connections might be able to give such guarantees. On the other hand, it may seem unrealistic to assume delivery guarantees for connections between two distant participants to be known when the hardware structure is inhomogeneous.

We first introduce the notion of *locally admissible* machines and collections, which are those machines and collections that demand only local timeouts.

Definition 6 (Locally Admissible Machines). *A machine* M *is called* locally admissible *if the following holds for all* $\mathsf{n} \in \Sigma^+$*:*

- *If M has a port* $\mathsf{fair\_snd\_n}?$*, then it also has the out-port* $\mathsf{n}!$*, but not the clock out-port* $\mathsf{n}^{\triangleleft}!$*.*
- *If M has a port* $\mathsf{fair\_rcv\_n}?$*, then it also has the in-port* $\mathsf{n}?$*, but not the clock out-port* $\mathsf{n}^{\triangleleft}!$*.*

A collection $\hat{M}$ or a structure $(\hat{M}, S)$ is locally admissible if every machine of $\hat{M}$ is locally admissible.

Definition 7 (Fair Schedulerwith Local Timeouts). *A machine* M *is called* fair with local timeouts *if* M *is fair, and if for all clock out-ports* $\mathsf{p}^{\triangleleft}!$ *of* M *there is a polynomially bounded function* $J_{\mathsf{p}^{\triangleleft}!} : \mathbb{N}_0 \to \mathbb{N}_{>0}$ *such that the following holds.*

[7] From "protocol π using primitive X implements primitive Y," and "protocol ρ implements primitive X" we could still conclude "protocol π using protocol ρ implements protocol π using primitive X. But from that we could not deduce "protocol π using protocol ρ implements primitive Y."

– *For any time port* $\mathsf{t}^{\triangleleft}!$ *and any closed collection* $\hat{C}$ *containing* M *the following holds with probability one over the runs of* $\hat{C}$*: The machine* M *does not schedule the port* $\mathsf{t}^{\triangleleft}!$ *more than* $J_{\mathsf{p}^{\triangleleft}!}(k)$ *times without scheduling* $\mathsf{p}^{\triangleleft}!$ *at least once.*
– *In its first activation,* A *writes* $J_{\mathsf{p}^{\triangleleft}!}(k)$ *(in unary representation) to* fair_snd_p! *and* fair_rcv_p! *(provided that these are ports of* A*).*

Based on this definition, fair reactive simulatability with local timeouts, written $\geq_{\mathsf{sec}}^{\mathsf{locrel}}$, is defined in the usual manner except that we additionally only allow configurations with locally admissible honest users. It should be remarked that the composition theorem still holds with the proof being conducted as for the previous extensions.

Similar to the notion of fairness with global timeouts, we neither require that fair_... is a port of M nor that M schedules that port (provided that it is a port of M). This is no weakness of the definition; in the first case, the machine possessing the port may schedule it, in the second case M is forced to eventually schedule that port since M is required to be fair.

Note that our notion of local timeouts allows the simulator to give different delivery guarantees for protocol-internal connections than the real adversary does. In particular, three repeaters implement two repeaters (cf. Figure 3) when considering local (in contrast to global) timeouts.

6 Relations Among the Notions

We finally investigate relations among the described notions and relations to the asynchronous scheduling model.

6.1 Non-Trivial Protocols

With fair reactive simulatability we have a security notion that allows us to capture the idea of protocols that eventually terminate. It is an interesting question whether requiring protocols to terminate will lessen the number of realizable cryptographic tasks. In other words, we ask whether there is a protocol task that can be realised with respect to non-trivial reactive simulatability, but not with respect to fair reactive simulatability.

The proof argument from [FLM86]—adapted to the case of fair reactive simulatability—shows that no protocol can be as secure as the task *broadcast* in the sense of fair reactive simulatability. Yet, for non-trivial reactive simulatability, a trivial protocol exists that is as secure as broadcast (it is an asynchronous variant of Protocol 1 in [GL02]).

We conclude without further proof:

Theorem 1. *There is no protocol that is as secure as broadcast in the sense of fair reactive simulatability. However, there is a protocol that is as secure as broadcast in the sense of non-trivial reactive simulatability.*

6.2 Drawing Profit from Timeouts

In this subsection we give a simple example of a protocol that can be securely realized in the sense of fair reactive simulatability with global timeouts, but is impossible in principle in a security model with only a fair adversary that does not provide timeouts. The protocol in question is the one that enables one machine M_1 to check whether another machine M_2 got input already.

Let us assume that the machines M_1 and M_2 communicate with each other through a secure connection. However, at least M_1 may be corrupted. Then, we are looking for a protocol that guarantees the following. Assume M_2 requests to check whether M_1 already got input. Then, if M_1

indeed got input at that point, M_2 shall eventually output that input. In any case, M_2 eventually outputs either M_1's input or $\perp$. The formal definition of the corresponding ideal protocol RNVP can be found in Appendix B.1.(RNVP stands for "rien ne va plus" to reflect that any input that the "player" M_1 has given so far is fixed by a signal of the "croupier" M_2.) Of course, we cannot expect that M_2 generates output immediately so that the specific scheduling influences what it means for M_2 to "eventually" generate output.

This protocol task can be securely realized when interacting with adversaries that are fair with global timeouts. Intuitively, the machine M_2 sends a request to M_1 and sets a timeout within which an answer from an uncorrupted machine must be received. The machine M_1 answers with its own input and the machine M_2 either outputs whatever it received from M_1 or outputs $\perp$ if no answer is received before the time out (in which case M_1 must be corrupted).

Theorem 2. *There is a real protocol that is as secure as* $(\mathsf{RNVP})_p$ *in the sense of fair reactive simulatability with global timeouts of delay* $p(k) := 3$*.*

Formal definitions and a sketch of the proof of this theorem are given in Appendix B.1.

However, no protocol exists that is as secure as RNVP in the sense of fair reactive simulatability (without timeouts) because the maximal delay times are not known to the uncorrupted protocol parties.

Theorem 3. *Any protocol with the communication structure and trust model described above is not as secure as* RNVP *in the sense of fair reactive simulatability.*

A sketch of the proof can be found in Appendix B.2.Note that the theorem statement would not be stronger if it considered *delayed* ideal protocols $(\mathsf{RNVP})_p$, since delay-boxes do not add any additional capabilities to the simulator if the considered adversaries do not provide timeouts. In fact, the proof sketch given in the appendix applies also to this seemingly stronger statement. The above example furthermore serves as a separating example for the notions of fair reactive simulatability with local timeouts on the one hand and fair reactive simulatability (without timeouts) on the other hand, with very similar proofs.

6.3 Global vs. Local Timeouts

It is reasonable to ask whether there are protocol tasks which actually need global timeouts (possibly with respect to delay boxes). More precisely, do there exist ideal protocols that can be securely realized when assuming fair adversaries with global timeouts but that cannot be securely realized in the presence of fair adversaries with local timeouts? We show that this question can be answered in the positive.

As an example of a protocol task for which one needs global delivery guarantees, suppose that a machine M_1 wants to send a k-bit message privately to another machine M_4. Let us assume that both machines are incorruptible and directly connected via an authenticated but insecure channel (e.g., a telephone wire). Furthermore, we assume that both machines can communicate securely only indirectly via two machines (e.g., servers) M_2 and M_3. Our goal is to find a protocol for $\mathsf{M}_1, \dots \mathsf{M}_4$ which achieves the following requirements:

Protocol task CW (Chinese Whisper)

1. If neither M_2 nor M_3 is corrupted, M_4 outputs M_1's input, and M_1's input stays secret.
2. If M_2 and/or M_3 is corrupted, M_4 outputs either M_1's input or an abort message; no secrecy is required.

To get a seperating protocol task, we would like to have information-theoretic security guarantees from the protocol; in particular this means that we consider statistical indistinguishability for the class of exponentially small functions below instead of the more common computational indistinguishability, cf. Definition 1. Actually, in face of a polynomially bounded adversary, public-key encryption over the authenticated channel between M_1 and M_4 would satisfy our needs. A natural and interesting question is whether there is also a protocol that distinguishes local and global timeouts w.r.t. computational security. We currently know of no such protocol.

An ideal structure reflecting these goals could consist of a trusted host $\mathsf{CW}_{\{1,\dots,4\}}$ with code (in the uncorrupted case) as follows:

Program of $\mathsf{CW}_{\{1,\dots,4\}}$

1. If activated with in?-input: set *data* to that input, then output 1 on loop!, and inform the adversary by sending it a 1 via public!. Ignore any further in?-inputs.
2. If activated with loop?-input (which can only happen after an in?-input, so *data* is set), output the value of *data* on out! and halt.

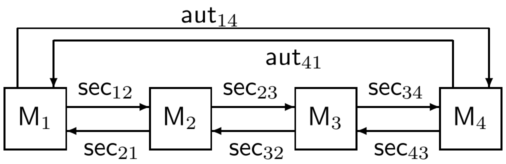


Fig. 4. All indicated connections are adversary-clocked. Each M_i may optionally have time_i?, time_i! ports and/or fair_...?, fair_...$^{\triangleleft}$! ports for local connections.

The service ports are in, out and public. In case of corrupted machines M_2 and M_3, one would of course modify this specification to send not only a notification "1", but instead the whole message *data* over public!: one can certainly not expect a message to be transmitted in a statistically indistinguishable way with corrupted intermediate hosts but without pre-distributed secrets. Furthermore, we allow a special abort message from the adversary which causes $\mathsf{CW}_{\{1,4\}}$ (the trusted host in the case of corrupted parties M_2 and M_3) to output $\perp$ instead of the actual message to be transferred. This models that we do not expect correct message delivery if M_2 and M_3 are corrupted; only eventual delivery of either $\perp$ or the correct message is mandatory.

The communication situation from our motivation above can be modeled by a structure structure $(\hat{M}^*, S^*)$ with machines $\hat{M}^* := \{\mathsf{M}_1, \mathsf{M}_2, \mathsf{M}_3, \mathsf{M}_4\}$ and service ports in and out. The only allowed connection between the machines are those depicted in Figure 4. All connections except the two between M_1 and M_4 (the telephone wire) are secure. We stress that we do not fix the actual protocol (i.e., the code) the machines $\mathsf{M}_1, \dots, \mathsf{M}_4$ run.

Theorem 4. *For any protocol with the communication structure and trust model as described above, it holds that the protocol is neither statistically as secure as* CW *in the sense of fair reactive simulatability, nor is it statistically as secure as* CW *in the sense of fair reactive simulatability with local timeouts. This holds independent of the code of the machines* $\mathsf{M}_1, \dots, \mathsf{M}_4$.

We give a proof sketch in Appendix B.3.Finally, we investigate the same protocol task with respect to global timeouts and delayed buffers. A proof sketch of the theorem is given in Appendix B.4.

Theorem 5. *There is a protocol that is statistically as secure as* $(\mathsf{CW})_p$ *in the sense of fair reactive simulatability with global timeouts for a sufficiently large delay polynomial* p.

7 Conclusions

The notion of simulatability has asserted its position as one of the fundamental concepts of modern cryptography. While this notion carefully captures that a distributed protocol does not behave any worse than an ideal specification, it however does not capture any form of liveness guarantees, i.e., that the protocol ensures that something good eventually happens. In particular, a protocol that does not create any output or that can be caused to hang indefinitely by corrupted parties serves as a good implementation of every ideal specification in the sense of reactive simulatability. In this paper, we investigated how one can extend the notion of reactive simulatability so that it additionally comprises liveness guarantees. The natural solutions and also the one we chose in this paper is to restrict the master scheduler—the adversary in our case—to fair scheduling, where notions of fairness that allow for reasoning about cryptography in a meaningful way still had to be defined.

To live up to the polynomial runtimes of the parties in cryptographic systems, we defined fairness as a polynomial-time variant of the usual fairness definition, i.e., we required that every message be scheduled within a specific, polynomially bounded number of steps of the adversary instead of requiring eventual delivery of every message. We further strengthened the definition by not only requiring that messages be delivered after some polynomial number of steps but by requiring that the number of steps, i.e., the maximum delay of messages, be made known explicitly to the protocol parties. We called this notion fairness with explicit timeouts, and we further distinguished variants with local and global timeouts.

We finally compared the resulting definitions of fair reactive simulatability, and we provided separating examples that helped to classify the strengths of the definitions. Somewhat counterintuitively, the examples have shown that protocols that are secure with respect to one definition might be insecure with respect to a definition that provides more comprehensive fairness guarantees. This stems from the fact that simulatability is defined by comparing a real protocol with an ideal specification, hence the more guarantees are given in the ideal model the more requirements have to be fulfilled by the real protocol.

An interesting research question for future work is the investigation of other notions of cryptography-suited fairness that reflect less abstract network models. Such notions could provide additional guarantees that are better suited for specific applications. Further research might also strive for conditions that are sufficient to prove that a protocol is as secure as an ideal functionality with respect to a given class of different fairness notions. For example, many protocols that are secure with respect to fairness with global timeouts, but do themselves not use timeouts, might be secure both with respect to fairness with and without timeouts, as well as possibly with respect to other notions in between.

Acknowledgements

This work was partially funded by the EC project ProSecCo under IST-2001-39227.

A Delayed Buffers

The problem with global timeouts that is sketched at the end of Section 5.1 is basically that guarantees given in some real and ideal structures have to be exactly identical. But then, the ideal adversary may not have enough freedom to adapt the response times of the ideal structure according to the response times of the real one. This can lead to strange effects as illustrated in the example from Section 5.1. The problem is of a general nature since we cannot expect all protocols for the same protocol task to have identical response times when assuming an identical network quality.

On the other hand, it is a tedious and possibly error-prone task to explicitly build another specific trusted host (or ideal structure) for each and every protocol which is to be proven secure; this would violate the idea of an abstraction from the considered class of real protocols. However, one can still start with a completely abstract specification of an ideal structure that is designed without taking care of, e.g., concrete response times of a real protocol. From that, one could construct a suitably delayed structure in a canonical manner so that the delayed structure can be securely realized by a specific real protocol. Thus, the only protocol-dependent parameter would be a specification of concrete extra delay times used as an adaptation of the initial ideal specification. If that construction is canonical enough, it will not break the abstractness of the ideal specification. In resemblance to the "shell constructs" of [HMQ04, Bac03], we therefore start with a structure $(\hat{M}, S)$, and replace it with a structure $\{(\hat{M}_p, S)\}_p$ that is parametrized with a function $p : \mathbb{N}_0^2 \to \mathbb{N}_0$. Intuitively, $p(J, k)$ indicates the factor with which the structure is delayed, where J is the delivery guarantee given by the adversary and k the security parameter.

Before going further, we need a tool for delaying connections. For a function $p : \mathbb{N}_0^2 \to \mathbb{N}_0$ and a connection name $\mathsf{n} \in \Sigma^+$, the *delay box* dbox_p_n is a machine with the ports as in Figure 5. The program of dbox_p_n is as follows:

Program of the delay box dbox_p_n

1. First, dbox_p_n waits to get a delivery guarantee $p(J, k)$ at port fair_n?.
2. Then, a counter *wait* is set to 1, and 1 is output on time_n!.
3. After that, on nonempty time_n?-input, *wait* is incremented, and again 1 is output on time_n!.
4. Every clk_n?-input c is forwarded to $\mathsf{n}^{\triangleleft}$!; if $c = 1$, then *wait* is reset to 0.
5. If at any time, $wait \geq p(J, k)$, then *wait* is reset and $\mathsf{n}^{\triangleleft}$! is clocked with 1.

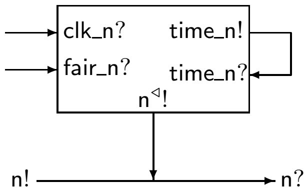


Fig. 5. A delay box dbox_p_n for a connection n. Unless indicated otherwise, all connections are adversary-clocked.

So essentially, dbox_p_n serves as a forwarder from clk_n? to the clock port $\mathsf{n}^{\triangleleft}$!. Since clk_n? is a simple in-port, an adversary that is fair with respect to global timeouts is not required to regularly give input to clk_n? (and thus schedule n). Merely, dbox_p_n itself ensures a regular scheduling of clk_n?. The function p and the delivery guarantee from the adversary determine how often clk_n? is scheduled at minimum.

Most of the time, it may seem reasonable to demand that p is polynomially bounded, i.e., that there is a bivariate polynomial q with $q(x, y) \geq p(x, y)$ for all $x, y \in \mathbb{N}_0$. Otherwise, not even a fair adversary with global timeouts guarantees that the connection n is scheduled regularly (i.e., at least once in a polynomial number of activations of the adversary). Delay boxes now allow for defining delayed structures:

Definition 8 (Delayed structures). *Let* $(\hat{M}, S)$ *be a structure and* $p : \mathbb{N}_0^2 \to \mathbb{N}_0$ *be a function. Then the* p-delayed structure $(\hat{M}_p, S)$ *is obtained by adding to* $\hat{M}$ *all machines* $\mathsf{dbox\_}p\mathsf{\_n}$ *for which* $\mathsf{n} \in \Sigma^+$ *begins with* $\mathsf{delay\_}$*, and* $\mathsf{n}?$ *or* $\mathsf{n}!$ *but not* $\mathsf{n}^{\triangleleft}!$ *is a port of* $\hat{M}$*.*

So essentially, $\mathsf{delay\_}\ldots$ connections enable the adversary to delay messages (polynomially) longer than it would be possible with a regular buffer. Loosely speaking, an adversary that is fair with global timeouts may schedule $\mathsf{delay\_}\ldots$ connections with delay $p(J, k)$.

We only delay connections that are explicitly labeled as $\mathsf{delay\_}\ldots$ in order to minimize the modification of the original structure. Note that when designing a protocol using a delayed trusted host we can either design the protocol without respect to the delay parameter; in that case the guarantees given on the $\mathsf{fair\_}\ldots?$ ports are useless for deriving delays *of that trusted host*. Or we can write the protocol in dependence of the concrete delay. Then a concrete realization of a delayed variant of the trusted host would imply a concrete delay polynomial, and we would know how to instantiate the larger protocol to allow for composition.

B Postponed Proof Sketches

This section contains more detailed definitions and sketches of proofs for the theorems above, which are not meant as complete proofs.

B.1 Details/Sketch of Proof for Theorem 2

Formally, the uncorrupted case can be modelled with a structure $(\mathsf{RNVP}_{\{1,2\}}, S_{\{1,2\}})$, where the service ports S are specified through their complements $S^c_{\{1,2\}} = \{\mathsf{in}!, \mathsf{in}^{\triangleleft}!, \mathsf{request}!, \mathsf{request}^{\triangleleft}!, \mathsf{out}?\}$, and the ports of the trusted host $\mathsf{RNVP}_{\{1,2\}}$ are $\mathsf{in}?, \mathsf{request}?$, $\mathsf{out}!$, $\mathsf{out}^{\triangleleft}!$, $\mathsf{commit}?$, $\mathsf{req\_info}!$, $\mathsf{req\_info}^{\triangleleft}!$, $\mathsf{in\_info}!$, $\mathsf{in\_info}^{\triangleleft}!$, $\mathsf{loop}!$, and $\mathsf{loop}?\}$. For $\mathsf{RNVP}_{\{1,2\}}$'s program, we initially assume $\mathit{data} = \bot$, $\mathit{com} = \mathsf{false}$. $\mathsf{RNVP}_{\{1,2\}}$ acts as follows:

Program of the trusted host $\mathsf{RNVP}_{\{1,2\}}$

1. On the first $\mathsf{in}?$ input $x \in \{0, 1\}$, and when $\mathit{com} = \mathsf{false}$, set $\mathit{data} := x$, and output and schedule x on $\mathsf{in\_info}!$;
2. on the first nonempty $\mathsf{request}?$ input, output 1 on $\mathsf{loop}!$, and output and schedule 1 on $\mathsf{req\_info}!$.
3. on nonempty $\mathsf{loop}?$ input, output and schedule data on $\mathsf{out}!$ and halt;
4. on nonempty $\mathsf{commit}?$ input, and when having received $\mathsf{request}?$ input, set $\mathit{com} := \mathsf{true}$.

So already in case of a fair scheduler, $\mathsf{RNVP}_{\{1,2\}}$ guarantees responses upon $\mathsf{request}?$-inputs.

A structure $(\mathsf{RNVP}_{\{2\}}, S_{\{2\}})$ for the case of a corrupted M_1 can be derived simply by setting $\mathsf{RNVP}_{\{2\}} := \mathsf{RNVP}_{\{1,2\}}$ and $S^c_{\{2\}} := S^c_{\{1,2\}} \setminus \{\mathsf{in}!, \mathsf{in}^{\triangleleft}!\}$. So in case of a corrupted M_1, the adversary gains control over the connection in. These two structures already form our ideal protocol RNVP.[8]

For realizing this system with a two-party protocol, we assume a structure $(\hat{M}_{\{1,2\}}, S_{\{1,2\}})$ with machines $\hat{M}_{\{1,2\}} = \{\mathsf{M}_1, \mathsf{M}_2\}$ and service ports $S_{\{1,2\}}$ as above in the uncorrupted case. The machines M_1 and M_2 have only one bidirectional and adversary-clocked secure channel in between them, modelled by two connections sec_{12} and sec_{21}.

The machine M_1 has the in-port $\mathsf{in}?$; M_2 has the in-port $\mathsf{request}?$, and the out-port $\mathsf{out}!$ (along with the corresponding clock out-port $\mathsf{out}^{\triangleleft}!$). Furthermore we allow, but do not mandate ports

[8] The case of a corrupted M_2 is not very interesting and omitted for simplicity. However, the statements below also hold for this case.

$\mathsf{time\_}i!, \mathsf{time\_}i?$, and/or $\mathsf{fair\_}i?, \mathsf{fair\_}i^{\triangleleft}!$ for the machines M_i $(i \in \{1,2\})$. Thus, the machines may at wish receive delivery guarantees (which are, of course, only meaningful in case of a timeout-fair adversary).

The structure $(\hat{M}_{\{2\}}, S_{\{2\}})$ for the case of a corrupted M_1 is derived canonically.

The ideal protocol RNVP just presented can be implemented when dealing with timeout-fair adversaries:

Theorem 6. *There is a real protocol of the form above, that securely implements* $(\mathsf{RNVP})_p$ *for the delay function* $p(k) := 3$.

Sketch of proof. To determine the protocol, it suffices to describe machines M_1 and M_2.

Only M_2, but not M_1, has the optional $\mathsf{time\_}$ and $\mathsf{fair\_}$ ports. M_1 answers the first sec_{21}-request from M_2 with the first in? input M_1 got, or $\perp$, if there was no input so far.

In its first activation, M_2 polls (via $\mathsf{fair\_2}^{\triangleleft}! := 1$) a delivery guarantee J. After that, when having got request? input, M_2 sends 1 over $\mathsf{sec}_{21}!$, and then waits J time_2?-activations (where time_2! is constantly fed with 1's) for a response on $\mathsf{sec}_{12}?$. If there is a response $x \in \{0,1\}$, M_2 out!-outputs and schedules x; otherwise, M_2 out!-outputs and schedules $\perp$.

A proof of security can be conducted as follows: one can transform a given real configuration step-wise into an ideal configuration. In each step, only a small modification is done, such that the view of the (unchanged) honest user is easily seen to remain unchanged. Now transitivity of simulatability can be used to derive indistinguishability of the user-views in real, resp. ideal configuration. If the ideal adversary does not depend on H, but only on the real adversary, this shows even universal simulatability.

Assume M_1 is not corrupted. We transform a given real configuration $(\hat{M}_{\{1,2\}}, S_{\{1,2\}}, \mathsf{H}, \mathsf{A})$ (with globally timeout-fair A) step-wise and without changing H's view into an ideal configuration $(\left(\mathsf{RNVP}_{\{1,2\}}\right)_p, S_{\{1,2\}}, \mathsf{H}, \mathsf{Sim})$.

First, A is modified to never output a value $\notin \{\epsilon, 1\}$ on $\mathsf{sec}_{12}^{\triangleleft}!$ and $\mathsf{sec}_{21}^{\triangleleft}!$.

Then, we rename A's $\mathsf{sec}_{12}^{\triangleleft}!$ and $\mathsf{sec}_{21}^{\triangleleft}!$ ports into loop! and commit!, respectively. We also add the respective ports $\mathsf{loop}^{\triangleleft}!$ and $\mathsf{commit}^{\triangleleft}!$, which are clocked with 1 whenever loop!, resp. commit! gets a nonempty output. Furthermore, we introduce a new machine CS (that will later become part of the trusted host) with ports $\{\mathsf{sec}_{12}^{\triangleleft}!, \mathsf{sec}_{21}^{\triangleleft}!, \mathsf{commit}?, \mathsf{loop}?\}$. CS only forwards commit? to $\mathsf{sec}_{21}^{\triangleleft}!$, and loop? to $\mathsf{sec}_{12}^{\triangleleft}!$.

Next, we add two ports req_info! and $\mathsf{req\_info}^{\triangleleft}!$ to M_2. The first time M_2 gets req? input, it outputs 1 on both req_info! and $\mathsf{req\_info}^{\triangleleft}!$ as a notification. Analogously, M_1 gets in_info! and $\mathsf{in\_info}^{\triangleleft}!$ ports, which are used to forward in? input. Furthermore, CS is changed to have req_info? and in_info? ports which are ignored.

Then, we change CS to ignore all inputs until the first req_info? input arrives. Next, CS is changed again to always output ϵ on $\mathsf{sec}_{12}^{\triangleleft}!$ until $\mathsf{sec}_{21}^{\triangleleft}!$ has been clocked. Finally, CS now halts after the first loop? input.

In the next step, the loop connection is turned into a self-loop of CS which A only clocks. Therefore, loop! is renamed into $\mathsf{loop}^{\triangleleft}!$, and CS outputs 1 on loop! upon every req_info? input.

Now we introduce a delay box dbox_3_loop and rename A's $\mathsf{loop}^{\triangleleft}!$ port to clk_loop!. A also gets an additional $\mathsf{clk\_loop}^{\triangleleft}!$ port that is 1 clocked whenever clk_loop! got output, or the original A would have 1-clocked $\mathsf{sec}_{12}^{\triangleleft}!$. A connects to the other free ports of dbox_3_loop in the obvious manner without losing its regained reliability property. Specifically, A can schedule $\mathsf{time\_loop}^{\triangleleft}!$ so that dbox_3_loop never 1-clocks $\mathsf{loop}^{\triangleleft}!$ on its own.

Finally, we combine M_1, M_2 and CS into a new machine $\overline{\mathsf{RNVP}}_{\{1,2\}}$; this combination can be formulated equivalently without internal sec_{ij} connections. Furthermore, $\overline{\mathsf{RNVP}}_{\{1,2\}}$ can—without changing H's view—be modified to generate out!-output on loop? input, even without prior commit? input. But then, $\overline{\mathsf{RNVP}}_{\{1,2\}}$ is nothing but a reformulation of $\mathsf{RNVP}_{\{1,2\}}$.

None of these steps changed the view of H. Furthermore, the construction of the globally reliable simulator Sim (the modified A) was independent of H, so the established simulatability is universal. The proof in the corrupted case follows the same lines (although of course the concrete construction of trusted host and simulator differs), and will not be given here. □

B.2 Sketch of Proof for Theorem 3

For the following sketch of the proof idea, this is not relevant. Assume for contradiction to the statement, that we have a real protocol π of the form above that securely implements RNVP with respect to fair adversaries.

In the configurations in this proof, we will exclusively consider a honest user $\tilde{\mathsf{H}}$ that first gives random input to M_1 and then lets M_2 check for M_1-input. For an efficiently computable function $J : \mathbb{N}_0 \to \mathbb{N}_0$, let A^J be a round-robin scheduler, with the exception that scheduling of the M_1-M_2-connections is slowed down by a factor of $J(k)$. Let $\mathsf{A}^J_{\mathsf{cor}}$ denote the analogous construction in case of a corrupted M_1 which discards all messages from and to M_2. For every polynomially bounded J, both A^J and $\mathsf{A}^J_{\mathsf{cor}}$ are fair and polynomial.

Let Sim^J, resp. $\mathsf{Sim}^J_{\mathsf{cor}}$ be the corresponding (fair) simulators for fair A^J, resp. $\mathsf{A}^J_{\mathsf{cor}}$. As H's views in real configurations are always nonempty, the same applies in ideal configurations with overwhelming probability.[9]

In an ideal configuration with any $\mathsf{Sim}^J_{\mathsf{cor}}$, there is $\mathsf{out}!$-output in a polynomial prefix of H's view by definition of $\mathsf{RNVP}_{\{2\}}$. By simulatability, M_2 thus generates $\mathsf{out}!$-output after a polynomial number of activations in a configuration with any $\mathsf{A}^J_{\mathsf{cor}}$, i.e., when not getting any $\mathsf{sec}_{12}?$ input at all. So there is a polynomial $F = F(k)$ independent of J, such that M_2 generates $\mathsf{out}?$-output after $F(k)$ activations when not getting $\mathsf{sec}_{12}?$ input.

Now the view of M_2 in a configuration with $\tilde{\mathsf{H}}$ and A^J, and M_2's view with $\tilde{\mathsf{H}}$ and $\mathsf{A}^J_{\mathsf{cor}}$ are identical until a connection between M_1 and M_2 gets scheduled. So even in the uncorrupted case, for a suitably chosen $J(k) > F(k)$ and real configurations with A^J, M_2 eventually generates $\mathsf{out}!$-output that is independent of M_1's input. But by specification of $\tilde{\mathsf{H}}$ and definition of $\mathsf{RNVP}_{\{1,2\}}$, there is always $\mathsf{out}!$-output which is identical to the former $\mathsf{in}?$-input in a polynomial prefix of H's view. This gives the desired contradiction. □

B.3 Sketch of Proof of Theorem 4

For contradiction, assume protocol machines $\mathsf{M}_1, \ldots, \mathsf{M}_4$ that do implement the ideal system with respect to locally reliable adversaries. Consider adversaries $\mathsf{A}^J_{\mathsf{cor}}$ (for computable $J : \mathbb{N}_0 \to \mathbb{N}_0$) which attack the protocol in case of (passively) corrupted M_2 and M_3. $\mathsf{A}^J_{\mathsf{cor}}$ schedules round-robin except for messages between M_2 and M_3 which are delayed for $J(k)$ rounds. Because M_2 and M_3 are corrupted and thus part of the adversary, $\mathsf{A}^J_{\mathsf{cor}}$ can be changed to be locally timeout-fair (without changing the view of any machine connected to $\mathsf{A}^J_{\mathsf{cor}}$).

So by assumption, there are locally timeout-fair simulators $\mathsf{Sim}^J_{\mathsf{cor}}$ attacking the trusted host $\mathsf{CW}_{\{1,4\}}$ and yielding a user-view indistinguishable from that with $\mathsf{A}^J_{\mathsf{cor}}$. Each $\mathsf{Sim}^J_{\mathsf{cor}}$ is fair, so the ideal (and thus also the real) system generates output after a polynomial number of user-activations; however, the specific polynomial may depend on J. Let $F(k)$ be smallest natural number such that the probability for the real protocol to generate output after less than $F(k)$ steps (in a configuration with $\mathsf{A}^{2^k}_{\mathsf{cor}}$) is $\geq 1/2$. Since the runtime of the real protocol is bounded by

[9] For readability, in the following we will often omit the phrase "with overwhelming probability" where it is clear.

a polynomial $p(k)$, $F(k)-1$ is also bounded by $p(k)$ by definition of F. Thus F is polynomially bounded. But by definition of F, the real protocol with adversary $\mathsf{A}^F_{\mathsf{cor}}$ generates output prior to the first scheduling of any M_2-M_3-connection with non-negligible probability.[10]

Since F is polynomially bounded, even an adversary A^F that does not corrupt any party but schedules like $\mathsf{A}^F_{\mathsf{cor}}$ is locally timeout-fair. Using simulatability again, we get that with non-negligible probability, the real protocol generates non-$\perp$ output without using a M_2-M_3-connection. So a message is transferred from M_1 to M_4 only by means of an authenticated channel. However, then we can construct an adversary that simply guesses the most likely message which corresponds to a given transcript of this authenticated channel. This adversary can be shown to be a successful guesser and thus breaks the security of the real protocol. Note that as the ideal specification does not use delivery guarantees, this proof also applies in the fair case. $\square$

B.4 Sketch of Proof of Theorem 5

We describe only the machines $\mathsf{M}_1, \ldots, \mathsf{M}_4$. Briefly, M_1, on in?-input $x \in \mathbb{F}_2^k$, randomly picks $A \xleftarrow{R} \mathbb{F}_2^{k\times k}$ and $b \xleftarrow{R} \mathbb{F}_2^k$. Then, M_1 writes (start) on aut_{14}! and $(x, Ax+b)$ on sec_{12}!. ($Ax+b$ is a message authentication code for x as described in [Sti95].) From then on, M_1 ignores all inputs except the next aut_{41}!-input from M_4. Specifically, M_1 writes (A,b) on aut_{14}! and halts when receiving the next nonempty aut_{41}!-input.

The machines M_2 and M_3 only relay the first nonempty sec_{12}?-input (resp. sec_{23}?-input) to sec_{23}! (resp. sec_{34}!).

M_4 has a fair_4? port (together with the corresponding clock port $\mathsf{fair}_4^{\triangleleft}$!) to get a global delivery guarantee. So in its first activation, M_4 clocks $\mathsf{fair}_4^{\triangleleft}$! with 1 to get a global guarantee $J \in \mathbb{N}_0$ at port $\mathsf{fair}_4^{\triangleleft}$?. Furthermore, on receiving (start) at aut_{14}? and $(x, \tilde{x}) \in \mathbb{F}_2^k \times \mathbb{F}_2^k$ at sec_{34}? (in any order), M_4 sends (ack) over aut_{41}! back to M_1. Then, on aut_{14}-input $(A,b) \in \mathbb{F}_2^{k\times k} \times \mathbb{F}_2^k$, M_4 generates and 1-clocks out!-output y. Here, $y = x$ if $Ax+b = \tilde{x}$, and $y = \perp$ otherwise. As a timeout mechanism, M_4 outputs (and 1-clocks) $\perp$ on port out! and halts when there are more than $5J+1$ nonempty $\mathsf{time\_loop}_4$-inputs after the initial (start) message.

The security of this real protocol can be proven by a step-wise transformation of a real configuration into an ideal one, where each step leaves the view of the honest user indistinguishable from the original one. Again, we only provide a quick overview of the proof. Into a given configuration in the uncorrupted case, we first insert the following machine CW' in between the honest user and the protocol machines with service ports (i.e., M_1 and M_4). CW' relays only the first in?-input to M_1, and only the first out!-output from M_4 which arrives after the first in?-input. Furthermore, this out!-output is substituted with the former in?-input.

Next, we change M_1 to replace the first in?-input x with 0^k and M_4 to replace all outputs by 1. After that, the machines A and $\mathsf{M}_1, \ldots, \mathsf{M}_4$ can be combined into a new machine Sim'. Finally, CW' and Sim' can simultaneously be changed into machines CW and Sim: CW is the trusted host $\mathsf{CW}_{\{1,\ldots,4\}}$, and Sim clocks CW's $\mathsf{loop}^{\triangleleft}$! port instead of writing to out!.

The timeout-fair property of Sim can then be restored by introducing a delay box $\mathsf{dbox\_}p\mathsf{\_loop}$ for the delay function $p(J,k) = 7J^2 + 2J$, and adapting Sim to this delay box in the obvious way. Intuitively, the construction of M_1 and M_4, and that of p allows Sim to schedule the ports of $\mathsf{dbox\_}p\mathsf{\_loop}$ in a timeout-fair manner, without ever forcing $\mathsf{dbox\_}p\mathsf{\_loop}$ to clock $\mathsf{loop}^{\triangleleft}$! on its own.

Note that this sketch constructed a simulator Sim which depends only on A, but not on H. The proof for the corrupted case is very similar, but additionally makes use of the properties of

[10] F may or may not be computable, but in any case can be approximated suitably.

the message authentication code to ensure the integrity of the transmitted message. We omit a sketch here. □

References

[ADG84] Chagit Attiya, Danny Dolev, and Joseph Gil. Asynchronous byzantine consensus. In *Third Annual ACM Symposium on Principles of Distributed Computing, Proceedings of PODC 1984*, pages 119–133. ACM Press, 1984.

[Bac03] Michael Backes. Unifying simulatability definitions in cryptographic systems under different timing assumptions. In Roberto Amadio and Denis Lugiez, editors, *Concurrency Theory, Proceedings of CONCUR 2003*, volume 2761 of *Lecture Notes in Computer Science*, pages 350–365. Springer-Verlag, 2003. Full version online available at `http://eprint.iacr.org/2003/114.ps`.

[BHMQU05] Michael Backes, Dennis Hofheinz, Jörn Müller-Quade, and Dominique Unruh. On fairness in simulatability-based cryptographic systems. In *3rd ACM Workshop on Formal Methods in Security Engineering: From Specifications to Code*, pages 13–22, September 2005.

[BOCG93] Michael Ben-Or, Ran Canetti, and Oded Goldreich. Asynchronous secure computation. In *Twenty-Fifth Annual ACM Symposium on Theory of Computing, Proceedings of STOC 1993*, pages 52–61. ACM Press, 1993.

[BPSW02] Michael Backes, Birgit Pfitzmann, Michael Steiner, and Michael Waidner. Polynomial fairness and liveness. In *15th IEEE Computer Security Foundations Workshop, Proceedings of CSFW 2002*, pages 160–174. IEEE Computer Society, 2002. Online available at `http://www.zurich.ibm.com/~mbc/papers/BPSW_02Liveness.ps`.

[BPW07] Michael Backes, Birgit Pfitzmann, and Michael Waidner. The reactive simulatability (RSIM) framework for asynchronous systems. To appear in *Information and Computation*, 2007. Preliminary version available online at `http://eprint.iacr.org/2004/082.ps`.

[BT85] Gabriel Bracha and Sam Toueg. Asynchronous consensus and broadcast protocols. *Journal of the ACM*, 32(4):824–840, 1985.

[Can00] Ran Canetti. Security and composition of multi-party cryptographic protocols. *Journal of Cryptology*, 3(1):143–202, 2000. Full version online available at `http://eprint.iacr.org/1998/018.ps`.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42th Annual Symposium on Foundations of Computer Science, Proceedings of FOCS 2001*, pages 136–145. IEEE Computer Society, 2001. Full version online available at `http://www.eccc.uni-trier.de/eccc-reports/2001/TR01-016/revisn01.ps`.

[CKL03] Ran Canetti, Eyal Kushilevitz, and Yehuda Lindell. On the limitations of universally composable two-party computation without set-up assumptions. In Eli Biham, editor, *Advances in Cryptology, Proceedings of EUROCRYPT '03*, volume 2656 of *Lecture Notes in Computer Science*, pages 68–86. Springer-Verlag, 2003. Full version online available at `http://eprint.iacr.org/2004/116.ps`.

[CKLS02] Christian Cachin, Klaus Kursawe, Anna Lysyanskaya, and Reto Strobl. Asynchronous verifiable secret sharing and proactive cryptosystems. In *9th ACM Conference on Computer and Communications Security, Proceedings of CCS 2002*, pages 88–97. ACM Press, 2002.

[CKPS01] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. In Joe Kilian, editor, *Advances in Cryptology, Proceedings of CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 524–541. Springer-Verlag, 2001.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *34th Annual ACM Symposium on Theory of Computing, Proceedings of STOC 2002*, pages 494–503. ACM Press, 2002. Extended abstract, full version online available at `http://eprint.iacr.org/2002/140.ps`.

[DLS88] Cynthia Dwork, Nancy A. Lynch, and Larry J. Stockmayer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323, 1988.

[FLM86] Michael J. Fischer, Nancy A. Lynch, and Michael Merritt. Easy impossibility proofs for distributed consensus problems. *Distributed Computing*, 1(1):26–39, 1986.

[FLP83] Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. Impossibility of distributed consensus with one faulty process. In *Second ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, Proceedings of PODS 1983*, pages 1–7. ACM Press, 1983.

[GL02] Shafi Goldwasser and Yehuda Lindell. Secure computation without agreement. In Dahlia Malkhi, editor, *Distributed Computing, 16th International Conference, DISC 2002, Toulouse, France, October 28-30, 2002 Proceedings*, volume 2508 of *Lecture Notes in Computer Science*, pages 17–32. Springer, 2002.

[GMY04] Juan A. Garay, Philip MacKenzie, and Ke Yang. Efficient and secure multi-party computation with faulty majority and complete fairness. IACR ePrint Archive, January 2004. Online available at `http://eprint.iacr.org/2004/009/`.

[HMQ04] Dennis Hofheinz and Jörn Müller-Quade. A synchronous model for multi-party computation and the incompleteness of oblivious transfer. In *Workshop on Foundations of Computer Security, Proceedings of FCS 2004*, 2004. Full version online available at `http://eprint.iacr.org/2004/016`.

[HMQU05] Dennis Hofheinz, Jörn Müller-Quade, and Dominique Unruh. Polynomial runtime in simulatability definitions. In *18th IEEE Computer Security Foundations Workshop, Proceedings of CSFW 2005*, pages 156–169. IEEE Computer Society, 2005. Online available at `http://iaks-www.ira.uka.de/home/unruh/publications/hofheinz05polynomial.html`.

[PSW00] Birgit Pfitzmann, Matthias Schunter, and Michael Waidner. Secure reactive systems. Technical Report RZ 3206, IBM Zurich Research Laboratory, 2000. Online available at `http://www.semper.org/sirene/publ/PfSW1_00ReactSimulIBM.ps.gz`.

[PW01] Birgit Pfitzmann and Michael Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *IEEE Symposium on Security and Privacy, Proceedings of SSP 2001*, pages 184–200. IEEE Computer Society, 2001. Full version online available at `http://eprint.iacr.org/2000/066.ps`.

[Sti95] Douglas R. Stinson. *Cryptography – Theory and Practice.* CRC Press, 1995.

[Yao82] Andrew Chi-Chih Yao. Theory and applications of trapdoor functions. In *23th Annual Symposium on Foundations of Computer Science, Proceedings of FOCS 1982*, pages 80–91. IEEE Computer Society, 1982.